TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol.19, No. 5, October 2021, pp. 1553~1564
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v19i5.19273

Analysis of frequent itemset generation based on trie data structure in Apriori algorithm

Ade Hodijah, Urip Teguh Setijohatmo
Informatics and Computer Engineering, Politeknik Negeri Bandung, Bandung, Indonesia

Article Info

Article history:

Received Dec 30, 2020
Revised Apr 10, 2021
Accepted Apr 22, 2021

Keywords:

Apriori
Hash-node calculation
Level pruning
Multi-thread
Triedata structure

ABSTRACT

Apriori is one technique of data mining association rules that aims to extract correlations between sets of items in the transaction database. The main problem with the Apriori algorithm is the process of scanning databases repeatedly to generate itemset candidates. This research examines the combination of pruning by using the trieapproach and multi-thread implementation in three algorithms to obtain frequent itemset. Trie is a data structure in the form of an ordered tree to store a set of strings where every node in the tree contains the same prefix. The use of a full combination trie (different from frequent pattern (FP) tree using links) allows the implementation of arrays and the hash calculation to achieve the addressing of itemset combination. In this research, the measure to get the address is called Hash-node calculation used to update support value. For these three alternatives, run time processing is analyzed based on the number of itemset combinations and transaction data at a certain minimum support value. The experimental results show that an algorithm thatexploits resource capabilities by applying multi-threadperforms almost seven times betterthanan algorithm implemented in single-thread in calculating hash-node. The fastest run time of the multi-thread approach is 43 minutes with 150-itemset combinations on 100,000 transaction data.

Corresponding Author:

Ade Hodijah
Informatics and Computer Engineering
Politeknik Negeri Bandung
Bandung, Indonesia
Email: adehodijah@jtk.polban.ac.id

1. INTRODUCTION

Several studies related to the application of the a priori algorithm have been carried out including for clickstream data analyzing [1], reasons finding [2], display item maximizing [3], ozone profiling [4]. The main problem of the Apriori algorithm lies in the process of generating itemset candidates that uses the repetition of the

database scanning as much (2^n-1)*(read external) times or (2^n-1)*(m/b block read), where
k: a sum of the item, m: a sum of a database record, b: block size; in the context of calculating support count [5]. It would carry out a database scanning as 102410 times for the sale of 100 items. An alternative to overcome the problem is to avoid the database scanning repeatedly, but only once in the process of updating the support count (for all new transactions that occur in a specific period). Using a linear list here using a trie data structure is to accommodate the database scanning to only be done once or (m/b).

This research's background is that there is still room for developing problem-solving to get frequent itemset by combining methods from the previous study and exploiting computing resources. The problem of getting a frequent itemset was first presented at [5], which is considered one of the most important contributions to the subject where an algorithm called Apriori greatly influences the community of data mining association rules. Many results of Apriori-based modification research have emerged (e.g., priority modeling, iteration reduction, wolf searching, reduction in scanning transaction) by [6]-[9], and in [10] the triedata structure is used because it consumes less memory than a linear list. After all, the same information is stored once. For example, there are 100 transactions that one item is in all dealings; storage of one item is only used once. Whereas in a linear list, the item is stored one for each transaction.

The study [10] proposed an algorithm for finding association rules, namely a priori algorithm that has been developed using the trie data structure stored in an array. Each edge in the tree contains a label and links to child nodes. Each node contains the itemset frequency with the edge being a member of that itemset. Itemset members consist of a set of edges from path node level 1 to the designated node. The root node contains the value 0 because there is no itemset pointed to by the root node. The itemset length can be seen at the depth node. Otherwise, research [11] presents a new approach in data separation by modifying the native a priori algorithm using a tree-based approach. This study presents an approach that helps find itemset that appears frequently which were constructed by finding 1-itemset first. However, this research also uses frequent itemset generated method, but in contrast to [11], we modified the next level as candidate itemset generation step in Apriori based on fulfillment status of the last frequent itemset combination (level-i), namely level pruning, so that the trie is used to save frequent itemset candidates. Other than that, we created the formula called Hash-node calculation to get an index of the n-itemset when adding the support count value of that frequent itemset, otherwise, they [11] built a tree and all of the n-itemset combination using the 1-itemset and it still has to go through several nodes starting from 1-itemset to n-itemset which searched for. However, the using of a tree-based data structure [10], [11] still used large memory by scanning the tree many times, so it needs to create an algorithm for finding the node or index of the tree that in this research namely Hash-node calculation and minimize the cost of I/O process by using a parallel algorithm that in this research called as mult-thread solution.

From another perspective, it is common to use multi-thread and computing power with multicore architecture [12] in supporting data processing. This raises the suspicion that multi-thread in [13] and [14] can produce better performance also in getting this frequent itemset, as well as a challenge on how to determine the best performance process architecture that is applied to which subprocesses as threads and how big is the increase for the best multi-thread architecture in a single server environment, where an experiment in multi-node server environment was proposed by [15], [16].

The process of calculating the value of support in the candidate sets generation in this problem is a bottleneck process, because this process will cause delays and queues to be forwarded to the next process. If you only use a standard a priori algorithm with a trie data structure it will still take a long time. Therefore, the multi-thread concept is applied in this study to minimize the time required for

processing the support candidate sets.For the thread model, this research applies thread in each sum of transaction data and not in each itemset combination like [17] and implements multi-thread not like [18], which evaluates in single-threaded performance.

2. RESEARCH METHOD

The research conducted is a quantitative research using experimental research techniques. The experiment is divided into two types, namely single-thread and multi-thread processes. The processing time run by each method is compared to be analyzed in terms of speed efficiency for adding the value of the support count candidate set. The processing time determines the quality of the method designed as well as the quality of the efficiency of the thread used with a data structure that has been specially designed by applying the trieconcept.

The number of transactions that consist dataset for the experiment is 100,000 transactions and 5 to 150 item variants.The dataset used is different for each experiment, because the data is generated randomly using a method. The method was specifically made in this study to generate data according to the needs of this research software. Table 1 contains examples of data generated for experimental needs with the number of 15 item variants.

To calculate the value of support count, transaction data is viewed one by one. If a subset of transactions is found, the support count value increased by 1. Trie not only stores candidates but also all itemset from the given transaction data. After the first database, the frequency for each item is obtained. Storing the support count and itemset values will increase the memory requirement a little, in exchange, this increases the speed of retrieving item appearances or the support count values of the itemset [19].

The objectives of conducting this experiment are:

- Knowing the time required to calculate the support count candidate value, and to find out whether the calculation of the supportcountcandidateset using multi-thread can run faster than a single-thread.
- Knowing the performance of the threads applied to the software to the available processors based on time, also knowing the number of threads and the efficient multi-thread model in the application of the candidate set calculation process.

A thread is like a small, lightweight process in process. A collection of threads is called a process [20]. Multi-thread can be managed independently, also dependently data-driven. Distribution of threads on multi-thread that are managed data-driven dependently only on the amount of data. Each data goes through all processes, then the process for n data is applied to a thread. Meanwhile, the distribution of threads in multi-thread is managed independently, namely in each process only [21]. Figure 1 shows this research method.

Table 1. Data to be used for experiments

T ID	Items
T1	{1, 4, 7, 8, 9, 10, 14}
T2	{2, 3, 4, 5, 6, 7, 8, 10, 13, 14}
T3	{4, 6}
...	...
T10000	{1, 4, 7, 9, 12, 13}

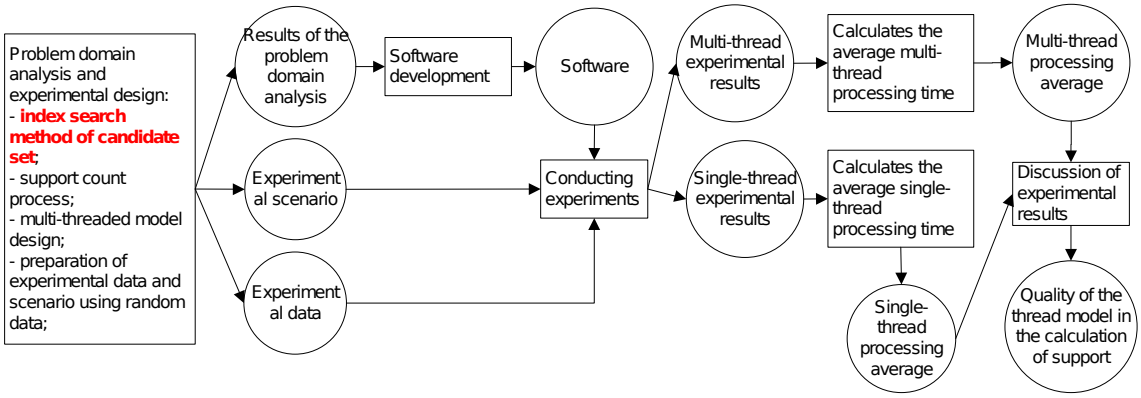

Figure 1. Research method

3. RESULTS AND ANALYSIS

In this research, the Apriori algorithm is developed based on a tree using the trie data structure approach, as seen in Figure 2. The modification of the Apriori algorithm is carried out based on the relationship between the parent value and the number of children to obtain a formula to calculate the Hash-node calculation. Table 2 explains how the trie data structure is used to determine the value of an index.

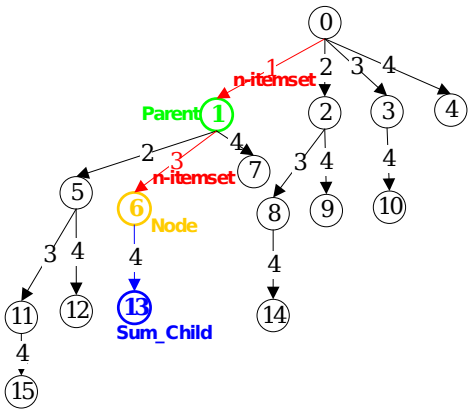

Figure 2. Illustration of trie for 4-itemset generation

Table 2. Illustration of n-itemset candidate generation

Node	Parent	Sum_Child	(n-Itemset)
0	-	4	0
1	0	3	1
2	0	2	2
3	0	1	3
4	0	0	4
5	1	2	1,2
6	1	1	1,3
7	1	0	1,4
8	2	1	2,3
9	2	0	2,4
10	3	0	3,4
11	5	1	1,2,3
12	5	0	1,2,4
13	6	0	1,3,4
14	8	0	2,3,4
15	11	0	1,2,3,4

3.1. Analaysis

The index in this research is not like a correlation between somethings [22], butit means an address of the candidate set (column Nodes) of each subset (column n-Itemset) generated in 4 item variant, as seen in Figure 2. The basic mechanism of the Apriori algorithm [5] is used in the reverse direction, where the database access is done first. The support count for each superset of the itemset candidate is calculated. So it takes the generation of a superset of itemset candidate as much as (2^n-1). For example, in the right column, namely, the n-itemset column shows a subset of a particular itemset that uses the formula. For example, n-itemset {1, 3} is in Node 6, and it has Parent which node is 1, and it has Sum_Child is 1 that is Node 13 which hasn-itemset {1, 3, 4}, as seen in Figure 3. The number of sets produced by each variant item 4 to 6 can be seen in the following figure.

len	k=1	k=2	k=3	k=4
	1	1,2	1,2,3	1,2,3,4
	2	1,3	1,2,4	
	3	1,4	1,3,4	
	4	2,3	2,3,4	
		2,4		
		3,4		
Sum	4	6	4	1

(a)

len	k=1	k=2	k=3	k=4	k=5
	1	1,2	1,2,3	1,2,3,4	1,2,3,4,5
	2	1,3	1,2,4	1,2,3,5	
	3	1,4	1,2,5	1,2,4,5	
	4	1,5	1,3,4	1,3,4,5	
	5	2,3	1,3,5	2,3,4,5	
		2,4	1,4,5		
		2,5	2,3,4		
		3,4	2,3,5		
		3,5	2,4,5		
		4,5	3,4,5		
Sum	5	10	10	5	1

(b)

len	k=1	k=2	k=3	k=4	k=5	k=6
	1	1,2	1,2,3	1,2,3,4	1,2,3,4,5	1,2,3,4,5,6
	2	1,3	1,2,4	1,2,3,5	1,2,3,4,6	
	3	1,4	1,2,5	1,2,3,6	1,2,3,5,6	
	4	1,5	1,2,6	1,2,4,5	1,2,4,5,6	
	5	1,6	1,3,4	1,2,4,6	1,3,4,5,6	
	6	2,3	1,3,5	1,2,5,6	2,3,4,5,6	
		2,4	1,3,6	1,3,4,5		
		2,5	1,4,5	1,3,4,6		
		2,6	1,4,6	1,3,5,6		
		3,4	1,5,6	1,4,5,6		
		3,5	2,3,4	2,3,4,5		
		3,6	2,3,5	2,3,4,6		
		4,5	2,3,6	2,3,5,6		
		4,6	2,4,5	2,4,5,6		
		5,6	2,4,6	3,4,5,6		
			2,5,6			
			3,4,5			
			3,4,6			
			3,5,6			
			4,5,6			
Sum	6	15	20	15	6	1

(c)

Figure 3. Candidate set for: (a) 4-itemset; (b) 5-itemset; and (c) 6-itemset

The pattern that results from the calculation of the number of candidate sets in Figure 1 is the Pascal triangle pattern [23] on n-itemset, where (a) n=4 is 4,6,4,1; (b) n=5 is 5, 10, 10, 5, 1; and (c) n=6 is 6, 15, 20, 15, 6, 1. The formula for getting the value of the Pascal triangle in the k-column of each n-itemset is used by the following formula:

$$C_{(n,k)}=\frac{n!}{k!(n-k)!} \quad (1)$$

The combination values obtained from each of these columns are then added up until k reaches len-1. Then the number of subsets can be calculated using the following formula:

$$y_1=\sum_{k=1}^{len-1}\frac{n!}{k!(n-k)!}(2)$$

Next is to trace the branch points (BP) for height/level starting from the 2-itemset (level=2) and its value is obtained from *len-2{...,len-2, i, j}*, beside that the value of BP is 0 (zero). The branch points itself contain nodes that produce a certain number

of subsets based on the number of itemset combinations. The number of branch point can be calculated using the $y_2$formula, which consists of two formulas of ($y_3 \wedge y_4$¿, as it can be seen in following formula:

$$y_2 = \sum y_3 \quad (3)$$

$y_2$is calculated by tracing the trieto the value of branch point that being searched from each of these columns and then added until k reaches *len* or level. The calculation for each value of the branches that is a subset can be calculated using the following formula:

$$y_3 = \frac{(n-(BP+1))(n-BP)}{2} \quad (4)$$

$y_3$is the total of nodes from the first node of all the sibling children nodes to the last node at the branch in the same parent node. After that, the value of y_3 must be reduced by the value of the older brother nodes in the same parent node that can be calculated using the following formula:

$$y_4 = \frac{(n-X_{len-1})(n-X_{len-1}+1)}{2} \quad (5)$$

The last is to trace the sequence of children in a subset of the branch points obtained. This value is obtained from the last 2 digits of the subset. For example, the subset value {1, 3, 4} is obtained from the calculation of 4-3, which is 1, where the subset form is changed in form {..., i, j}. Here is the formula for calculating the value of the sequence of children:

$$y_5 = j - i \quad (6)$$

All the equations that have been obtained are then added up to find the final index of the subset. To get the final index of an itemset, the calculations that must be done are:

$$index = y_1 + y_2 - y_4 + y_5 \quad (7)$$

3.2. Design

Trie of the candidate set is a trie tree which is all supersets of the itemset that represent all combinations of items sold. The trie here is used to calculate the support count of all itemset combinations. Suppose there are 4 item sets, the superset is all possible subsets, for example:

- 1-itemset:{1}, {2}, {3}, {4};
- 2-itemset:{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4};
- 3-itemset:{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4};
- 4-itemset:{1, 2, 3, 4},

Totaling 15 or 2^n-1 with n=4. This is a complete combination patterned, so that to achieve the address of a certain subset a formula can be formed.

Following is an illustration of calculating node values for 3-itemset combinations, for example, {2, 3, 4} with total itemset combinations is 4, as shown in Figure 4. An example of using Hash-node calculation to get the index value for 3-itemset {2, 3, 4} as illustrated in Figure 4, where BP=1 and 2, i=3, j=4, *len* or level=3 is as explained in detail at Table 3 and Table 4. The other example of Hash-node calculation for 4-

itemset {1, 2, 3, 4} as illustrated in Figure 4, where BP=1, i=3, j=4, *len* or level=4, so the index value is 15, and for 2-itemset {3, 4} is 10 with BP=0.

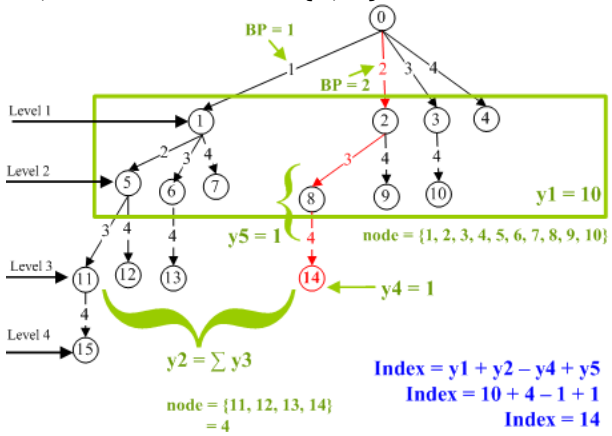

Figure 4. Hash-node calculation

Table 3. An example of calculating the y_1 and y_2 value for the subset {2, 3, 4} using Hash-node calculation

y_1		y_2		
$y_1=\sum_{k=1}^{len-1} \frac{n}{k!(n$		$y_2=\sum y_3$		
$y_1=\sum_{k=1}^{2} \frac{4!}{1!(4-$	+ $\sum_{k=2}^{2} \frac{4!}{2!(4-$	$y_3=\frac{(n-(BP+1))(n-}{2}$		
$y_1=\sum_{k=1}^{2} \frac{4!}{1!(3)!}$	+ $\sum_{k=2}^{2} \frac{4!}{2!(2)!}$	$y_3=\frac{(4-(1+1))(4-1}{2}$	+	$y_3=\frac{(4-(2+1))(4}{2}$
$y_1=\sum_{k=1}^{2} \frac{4*3*2}{1(3*2}$	+ $\sum_{k=2}^{2} \frac{4*3*2}{2*1(2}$	$y_3=\frac{(4-(2))(3)}{2}$	+	$y_3=\frac{(4-(3))(2)}{2}$
$y_1=\sum_{k=1}^{2} \frac{24}{6}$	+ $\sum_{k=2}^{2} \frac{24}{4}$	$y_3=\frac{2*3}{2}$	+	$y_3=\frac{1*2}{2}$
$y_1=4$	+ 6	$y_3=3$	+	$y_3=1$
$y_1=10$		$y_3=4$		
		$y_2=4$		

Table 4. An example of calculating the y_4, y_5, and index value for the subset {2, 3, 4} using Hash-node calculation

y_4	y_5	Index
$y_4=\frac{(n-X_{len-1})(n-X_{le}}{2}$	$y_5=j-i$	$Index=y_1+y_2-y$
$y_4=\frac{(4-3)(4-3+1)}{2}$	$y_5=4-$	$Index=10+4-1$
$y_4=\frac{1*2}{2}$	$y_5=1$	$Index=14$
$y_4=1$		

Data structures, as seen in Figure 5, are arrays with attributes consist of :

- Node as an index value of Hash-node calculation result.
- Data as an array of n-itemset combinations, such as 3-itemset {2, 3, 4}. The total combinations for 4 is (2^n-1), that is 15, as seen in Figure 4.
- Support isthevalue of support count. The default for each data is 0 (zero) and it will be counted plus one if it existed in data, and so on.

```
public class ItemsetNode {
    public int node;
    public int support;
    List<Integer> data = new ArrayList<>();
```

Figure 5. Data structure for n-Itemsettrieof candidate

After the index value is obtained, next is how to calculate the support value for each subset by adopting pruning from Apriori. The generation of the itemset candidate is done per-level. Only itemset candidates who meet the minimum support (frequent) are formed. Then find the candidate itemset's position in the trie of the candidate set to increase the support count. The steps for the Alternative_1 algorithmare:

- Establishing a trie of the candidate set as much (2n-1), where k is the number of itemsets sold.
- Forming a linear list of databases and superset of itemset for each transaction by combining all itemset purchased (transaction), which is as much as (2n-1) where ni is the number of itemsets purchased in the transaction of i.
- For each transaction, all subsets of the superset are searched for a position in the candidate set's trie to increase the support count.

The process model of the Alternative_1 algorithm is shown in Figure 6.

The difference between Alternative_1 and Alternative_2 algorithms is the generation of n-itemset. Suppose the construction of n-itemset in Alternative_1 is to combine all itemset and produce itemset of (2^n-1), where ni is the number of itemsets in transaction i. However, this is not the case with Alternative_2. Pruning is done by eliminating [24], [25] the previous long itemset whose frequency is below the minimum support thresholds (infrequent), as seen in Figure 7.

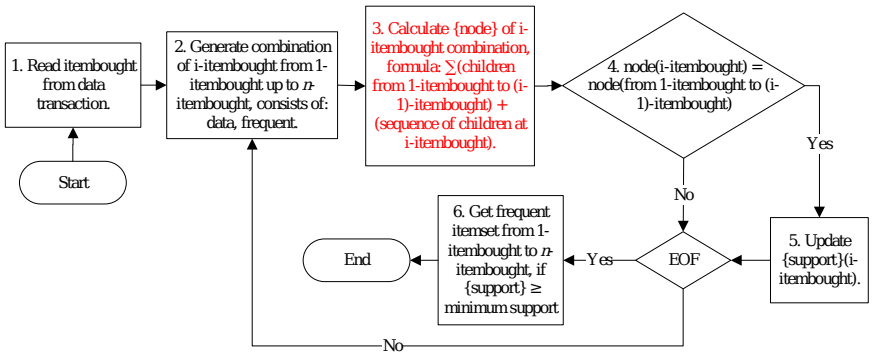

Figure 6. The process model of Alternative_1 algorithm

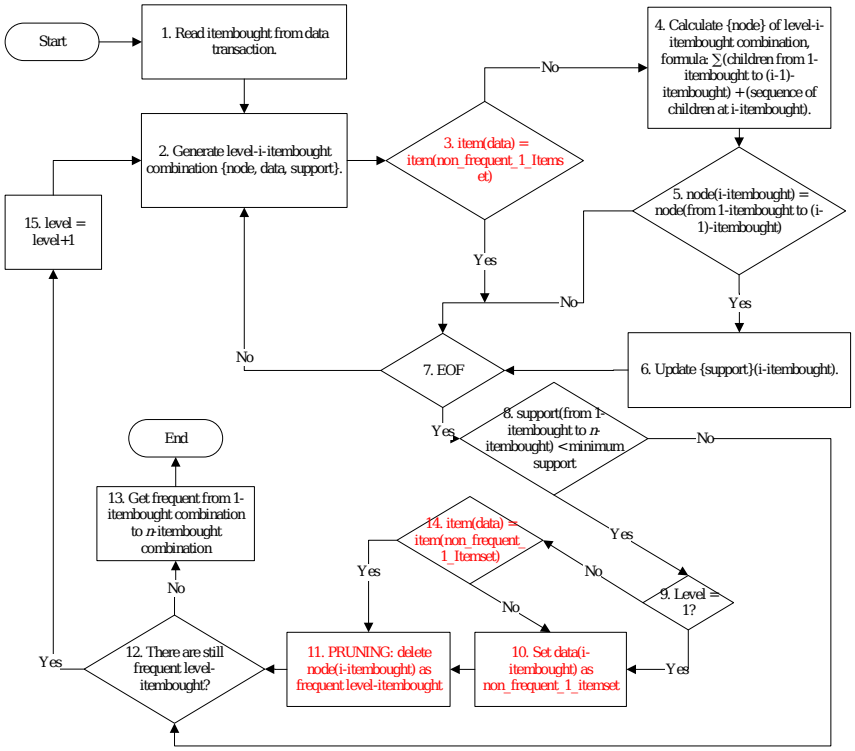

Figure 7. The process model of Alternative_2 algorithm

Alternative_3 treats subprocesses-2 and subprocesses-3 of Alternative_2 as threads by installing concurrent access control to avoid race conditions fighting over the value of support count at the same position in the trie. Multi-thread applied in this research used dependency data-driven [21]. Transaction data is divided into 4 threads and 8 threads. A different thread processes in this research based on the sum of the transaction data that can be seen in Figure 8 (a), so that race conditions may occur which is illustrated in Figure 8 (b), consisting of 4 threads.

To overcome race condition when update value of SupportCount, mutual exclusion is needed to access the same address of itemset {2, 3} as one of itemset combination in node 8 {2, 3}, node 11 {1, 2, 3}, node 14 {2, 3, 4}, and node 15 {1, 2, 3, 4} as seen in Table 1. Different threads process it without discarding one of the threads as collision handling [26] when two threads attempt to insert two different subtrees at the same location. In this research, the mutual exclusion mechanism uses a monitor using synchronized constructs when updating support values, as seen in Figure 9.

Threads in this research do not place on each itemset combination as it is unique. For example, itemset {1, 3, 4} produces 7 combinations are {1}, {3}, {4}, {1, 3}, {1, 4}, {3, 4}, {1, 3, 4}, so that threads is placed on sum of transaction database which has been divided by 4 and 8 threads. The experiment shows that the itemset combination threads method has a longer run time than the transaction data threads method.

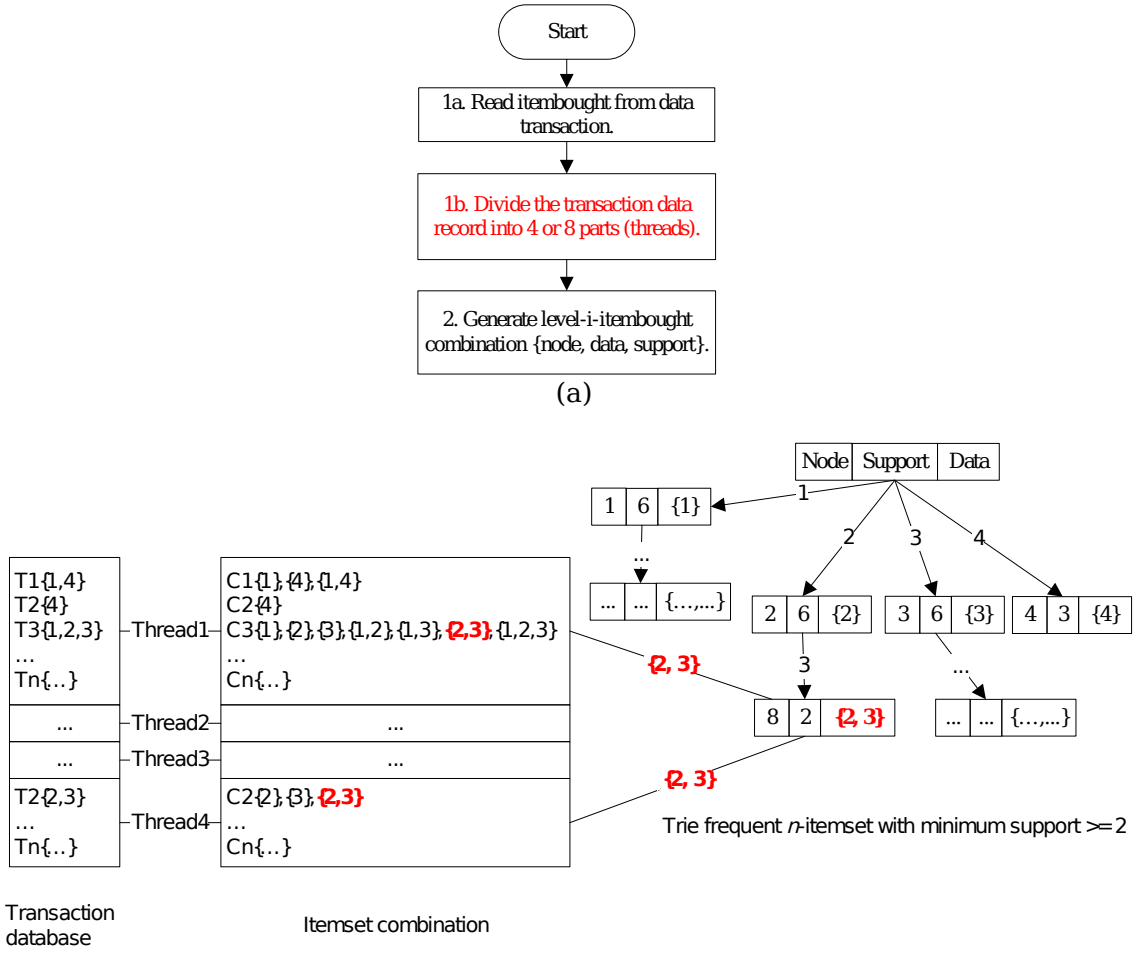

Figure 8. Process model of Alternative_3 algorithm with the example of the distribution of transaction data (a) into 4 threads (b)

```
for (int j = 0; j < indexList.size(); j++) {
    if( indexList.get(j) >= 0 ) {
        int nodeTrie = indexList.get(j);
        int supportCount = listCombinatorial.get(nodeTrie-1).getSupport();
        supportCount = supportCount + 1;

        synchronized (listCombinatorial) {
            listCombinatorial.get(nodeTrie-1).setSupport(supportCount);
        }
    }
}
```

Figure 9. Updating support value algorithm (simplified)

3.2. Experiments

The run time of generating frequent itemset of each proposed algorithm in this research, a test in a single server environment using a PC with Intel (R) Core (TM) specifications i5-8250U CPU @ 1.60GHz 1.80 GHz uses 12.0 GB RAM and uses the operating system Microsoft Windows 10. Test data obtained from generating random itemset combinations by a simple application representing the sales transaction data set. In getting the accurate run time results, testing is carried out 5 times for each minimum support value.

There are two types of the proposed algorithm, namely Single-thread processing average consists of Alternative_1 and Alternative_2; multi-thread processing average consists of Alternative_3 with 4, 8, and 20 threads. The experiment was carried out with 5experimental scenarios for each type of the proposed algorithm made based on the 3 alternatives mentioned previously, each of which is described as follows:

- S1 (Alternative_1): Processing the tries candidate set with one main process. Scenario-1 is intended to understand the workings and performance of tries.
- S2(Alternative_2): Processing with one main process like scenario S1 but added pruning that works per-level tree height, namely LevelPruning. This scenario intends to measure whether there is an increase in performance with pruning.
- S3 (Alternative_3 with 4 threads): Processing Alternative_2 with the use of the 4 threads. Scenario-3 intends to determine the performance improvement of tries with pruning and 4 threads to optimize CPU work due to idle time I/O.
- S4 (Alternative_3 with 8 threads): Processing Alternative_2 with the use of the 8 threads. Scenario-4 intends to determine whether there is an increase in performance with threads compared to S3.
- S5 (Alternative_3 with 20 threads): Processing Alternative_2 with the use of the 20 threads. Scenario-5 intends to determine whether there is an increase in performance with threads compared to S3 and S4.

To get the quality of the thread model in the calculation, each proposed algorithm above tested by 100,000 transaction data with six variations of n-itemset as Experimental data based on the Research Method. The variation number of itemset consists of 5-itemset, 25-itemset, 50-itemset, and 150 itemsets with the threshold of support value is 25% from the total of transaction data.

Table 5. Experimental results for 100,000 transaction data, 5-itemsetup to150-itemset, 25% minimum support in hours:minutes:seconds:milliseconds

Id	5-items	25-items	50-items	75-items	100-items	150-items
S1	00:00:00:906	X	X	X	X	X
S2	00:00:00:361	00:00:09:647	00:01:20:484	00:05:50:748	00:16:19:50	05:04:14:932
S3	00:00:00:161	00:00:03:368	00:00:31:216	00:01:55:805	00:09:05:147	02:00:55:783
S4	00:00:00:205	00:00:04:595	00:00:27:16	00:01:37:707	00:06:43:385	00:43:50:527
S5	00:00:00:242	00:00:11:888	00:04:34:563	00:19:43:225	01:09:36:369	05:54:35:530

The variation of the test data shown in Table 5 that the run time is the latest for the number of 5-itemset and 25% of minimum support is Alternative_1. It happens because before carrying out the counting node process, it takes a full itemset combination generation process for all n-itemset combinations, which is as much as (2^n-1) so that at n>5-itemset takes the longest run time processing until the process results in a hang. Therefore pruning operations are needed in this research using level pruning. If the result of the generation of a combination (level-1)-itemset meets the minimum support thresholds, then the itemset will be a candidate itemset at the next level. Level pruning's existence in generating itemset combinations based on this trie tree-level can save memory usage. The item matching process for each transaction record is not done for all itemset combinations (n-itemset), such as the proposed algorithm in Alternative_2 and Alternative_3.

The results from the 5-itemset up to 150-itemset of the three algorithms can be seen in the graph below with the value is converted from hours to milliseconds. The horizontal axis is the number of itemsets, and the vertical axis is run time processing (in millisecond), where red is for Alternative_1, green is for

Alternative_2, and blue is for Alternative_3 with 4 threads, yellow is for Alternative_3 with 8 threads, and orange is for Alternative_3 with 20 threads, as seen in Figure 10.

The fastest run time is 00:43:50:527 (2,630,527 msec) can be done by an algorithm from Alternative_3 with 8 threads for the number of 150-itemset and 25% of minimum support on 100,000 transaction data, while the run time generated by Alternative_2 is 05:04:14:932 (18,254,932 msec), and Altervative_1 itself is in hang that only seen in 5-itemset, as seen in Figure 10 (a), and it disappeared from Figures 10 (b) to 10 (f). Using multi-thread affects increasing run time that it is about 7 times faster than using no thread (Alternative_1 and Alternative_2). However, its run time performance does not in line with the number of multi-thread, as seen in Table 2. Alternative_3 with 4 thread (S3) is faster than 8 thread (S4) for the number of 5-itemset and 25-itemset, as seen in Figures 10 (a) and 10 (b). In contrast, Alternative_3 with 8 threads (S4) is faster than 4 threads (S3) for the number from 50-itemset to 150-itemset. Otherwise, Alternative_3 with 20 threads is the lastest from 5-itemset to 150-itemset, so the more threads did not affect to get the betterperformance. Figure 11 shows that the greater the number of the item variants, then the greater the time difference was obtained, but the greater the number of the threads did not always getting the faster the time of processing.

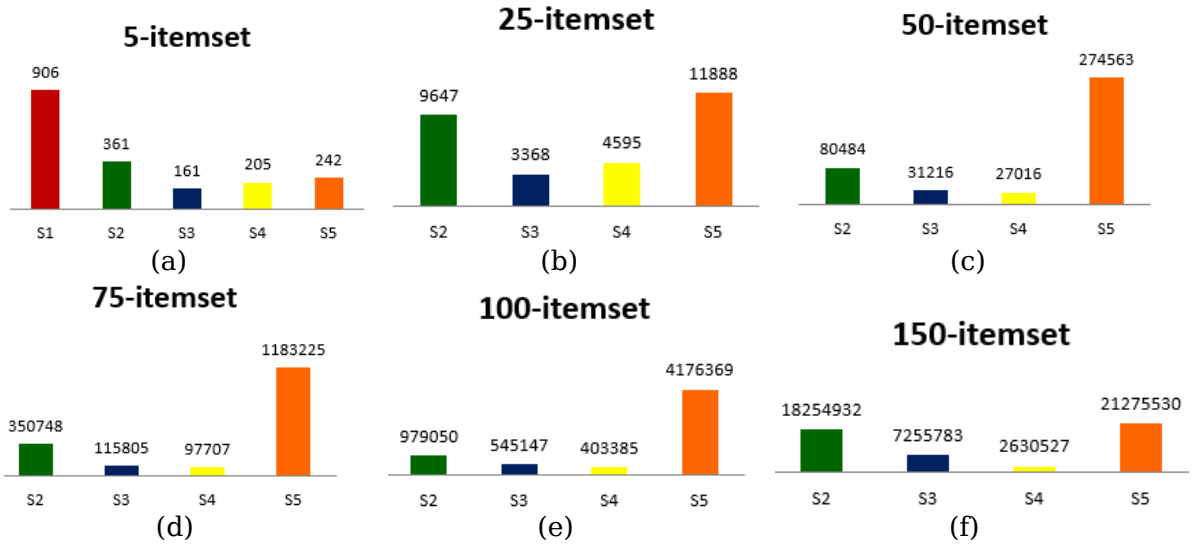

Figure 10. Experimental results for 100,000 transaction data in milliseconds: (a) 5-itemset, (b) 25-itemset,
(c) 50-itemset, (d) 75-itemset, (e) 100-itemset, and (f) 150-itemset with 25% of minimum support

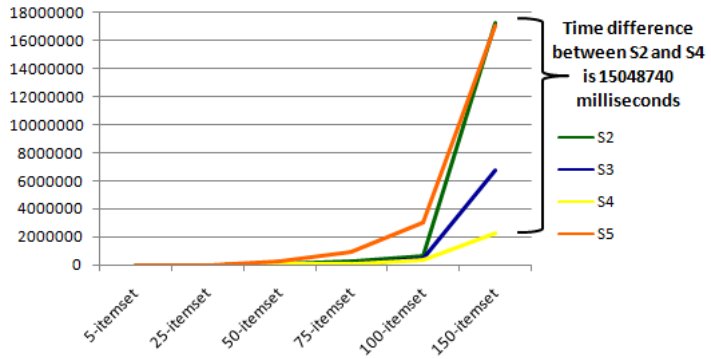

Figure 11. The difference in the processing time

The discussion of the experimental results in Table 4, it shows that the average time required for processing using multi-thread is faster than the single-thread model, despite for Alternative_3 with 20 threads (S5). The difference in experimental time is very small for item variants under 100-itemset and it increased dramatically over 100-itemset. However, the addition of the itemset variant to 150-itemset has resulted in almost 8 times the difference in processing time between S2 and S4, which is about 15048740 msec or about 4 hours. In the case of market basket analysis, the number of item variants has a big role in processing time where the number of candidate sets obtained is (2^n-1).

4. CONCLUSION

The Hash-node calculation as a current research's achievement is proposed three variants algorithm which has been able to generate frequent itemset with the fastest execution is the algorithm of Alternative_3that is 43 minutes 50 seconds 527 milliseconds with the number of multi-thread is 8 threads. Other than that, the experiment time between multi-thread and single-thread was very different at the 150-itemset, wherethe smallest time is multi-thread and the largest is single-thread.

There are two factors that are recommended to be further explored from the achievements of this research. First, the kind of other thread models (lightweight process/LWP), where the thread processing in this research is carried out based on the distribution (data-driven) of the number of transaction data in some threads. Second, the other approach to find the order of nodes in a trieat a certain level using the formula of ($y3 \wedge y4$¿, where the search based on the value of BP carried out in this research requires a recursive algorithm that requires a large enough memory, so that the available heap space is only a little. In addition, the experimental data used in this research are relatively small. Therefore, it is possible that the processing time needed is longer if the experiment uses real data in supermarkets with millions of item variants, and the number of transactions reaching billions. Besides that, the development of this research is related to the need for a multi-thread processing of a real environment, such as through MapReduce-based frequent itemset mining algorithms.

REFERENCES

[1] Supriyadi, Y. Nurhadryani, and A. I. Suroso, “Website content analysis using clickstream data and apriori algorithm,” *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, vol. 16, no. 5, pp. 2118-2126, 2018, doi: 10.12928/TELKOMNIKA.v16i5.7573.

[2] A. H. Nasyuha, J. Jama, R. Abdullah, and Y. Syahra, “Frequent pattern growth algorithm for maximizing display items,” *TELKOMNIKA Telecommunication, Computing, Electronics and Control,* vol. 19, no. 2, pp. 390-396, 2021, doi: 10.12928/TELKOMNIKA.v19i2.16192.

[3] M. A. Abdulhamed, H. I. Mustafa, and Z. I. Othman, “A hybrid analysis model supported by machine learning algorithm and multiple linear regression to find reasons for unemployment of programmers in Iraq,” *TELKOMNIKA Telecommunication, Computing, Electronics and Control,* vol. 19, no. 2, pp. 444-453, 2021, doi: 10.12928/TELKOMNIKA.v19i2.16738.

[4] M. S. Johnson *et al.*, “Evaluation of potential sources of a priori ozone profiles for TEMPO tropospheric ozone retrievals,” *Atmos. Meas. Tech.*, vol. 11, no. 6, pp. 3457-3477, 2018, doi: 10.5194/amt-11-3457-2018.

[5] R. Agrawal, T. Imieliński, and A. Swami, “Mining Association Rules Between Sets of Items in Large Databases,” *ACM SIGMOD Rec.*, 1993, doi: 10.1145/170036.170072.

[6] R. Christopher and M. Shoerio, “Priority Model of Apriori Algorithm,” *Int. J. Adv. Res. Comput. Sci. Manag. Stud.,* vol. 3, no. 6, pp. 247-255, 2015.

[7] S. Rathod and A. Sharma, “Survey Paper for Enhancement of Apriori Algorithm,” *Int. J. Eng. Manag. Res. Page Number*, no. 6, pp. 808-811, 2016.

[8] G. Jain and D. Maurya, “Extraction of association rule mining using apriori algorithm with wolf search optimisation in R programming,” *Int. J. Recent Technol. Eng.*, vol. 8, no. 2 Special Issue 7, pp. 504-507, 2019,

doi: 10.35940/ijrte.B1094.0782S719.
[9] X. Yuan, "An improved Apriori algorithm for mining association rules," *AIP Conf. Proc.*, vol. 1820, no. 11, pp. 6521-6526, 2017, doi: 10.1063/1.4977361.
[10] C. Mewada and R. Morena, "Trie Based Improved Apriori Algorithm to Generate Association Rules," *Int. J. Adv. Res. Comput. Commun. Eng. ISO 32972007 Certif.,*vol. 5, no. 10, pp. 430-436, 2016, doi: 10.17148/IJARCCE.2016.51087.
[11] A. Sarkar, A. Paul, S. K. Mahata, and D. Kumar, "Modified Apriori Algorithm to find out Association Rules using Tree based Modified Apriori Algorithm to find out Association Rules using Tree based Approach," *Int. J. Comput. Appl.*, no. October, 2017.
[12] A. T, S. Pasad R, and S. N. Tirumalarao, "Performance Evaluation of Apriori on Dual Core with Multiple Threads," *Int. J. Comput. Appl.*, vol. 50, no. 16, pp. 9-16, 2012, doi: 10.5120/7853-1088.
[13] Y. K. Abu Samra and A. Y. A. Maghari, "Enhancing FP-Growth Performance Using Multi-threading based on Comparative Study," *Int. J. Intell. Comput. Res.*, vol. 6, no. 3, pp. 613-620, 2015, doi: 10.20533/ijicr.2042.4655.2015.0076.
[14] S. Kumar and K. K. Mohbey, "A review on big data based parallel and distributed approaches of pattern mining," *J. King Saud Univ. - Comput. Inf. Sci.*, 2019, doi: 10.1016/j.jksuci.2019.09.006.
[15] S. Singh, R. Garg, and P. K. Mishra, "Performance Analysis of Apriori Algorithm with Different Data Structures on Hadoop Cluster," *Int. J. Comput. Appl.*, vol. 128, no. 9, pp. 45-51, 2015, doi: 10.5120/ijca2015906632.
[16] P. Singh, S. Singh, P. K. Mishra, and R. Garg, "A data structure perspective to the RDD-based Apriori algorithm on Spark," *Int. J. Inf. Technol.*, no. August, 2019, doi: 10.1007/s41870-019-00337-3.
[17] H. Zhian, M. Bayat, M. Amiri, and M. Sabaei, "Parallel processing priority trie-based IP lookup approach," *2014 7th Int. Symp. Telecommun. IST 2014*, 2014, dpp. 635-640, oi: 10.1109/ISTEL.2014.7000782.
[18] R. Binna, E. Zangerle, M. Pichl, G. Specht, and V. Leis, "a Height Optimized Trie Tree," *Proc. 2018 Int. Conf. Manag. Data - SIGMOD '18*, 2018, pp. 521-534, [Online]. Available: http://dl.acm.org/citation.cfm?doid=3183713.3196896.
[19] F. Bodon, "A Fast Apriori Implementation Hungarian Academy of Sciences," *Fimi*, vol. 3, p. 63, 2011, [Online]. Available: http://www.cs.bme.hu/~bodon/kozos/papers/bodon_trie.pdf.
[20] GeeksforGeeks, "Multithreading In Operating System," *GeeksforGeeks*, 2021. https://www.geeksforgeeks.org/multithreading-in-operating-system/ (accessed Jan. 19, 2021).
[21] Study Tonight., "What are Threads?," *Study Tonight.*, 2021. https://www.studytonight.com/operating-system/multithreading (accessed Jan. 19, 2021).
[22] J. Gibergans-Báguena, C. Hervada-Sala, and E. Jarauta-Bragulat, "The quality of urban air in Barcelona: A new approach applying compositional data analysis methods," *Emerg. Sci. J.*, vol. 4, no. 2, pp. 113-121, 2020, doi: 10.28991/esj-2020-01215.
[23] X. Gao and Y. Deng, "The pseudo-pascal triangle of maximum deng entropy," *Int. J. Comput. Commun. Control*, vol. 15, no. 1, pp. 1-10, 2020, doi: 10.15837/3735/ijccc.2020.1.3735.
[24] H. Bathla, "Apriori Algorithm and Filtered Associator in Association Rule Mining," vol. 4, no. 6, pp. 299-306, 2015.
[25] S. Dasgupta and S. Karmakar, "Food recommendation using classifier and modified Apriori algorithm," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 12, pp. 3967–3970, 2019, doi: 10.35940/ijitee.L3471.1081219.
[26] H. Jordan, P. Suboti, D. Zhao, and B. Scholz, "Brie: A specialized trie for concurrent datalog," *Proc. 10th Int. Work. Program. Model. Appl. Multicores Manycores, PMAM 2019*, pp. 31-40, 2019, doi: 10.1145/3303084.3309490.

BIOGRAPHIES OF AUTHORS

Ade Hodijah, obtained the Master degree in 2012 at InstitutTeknologi Bandung. She is lecturer of informatics and computer engineering at State Polytechnic of Bandung. Her main research interests is related to knowledge discovery.

Urip Teguh Setijohatmo, obtained the bachelor degree in 1990 at University of Kentucky and the Master degree in 2000 at Universitas Indonesia. He is lecturer of informatics and computer engineering of State Polytechnic of Bandung. His main research interests is in data engineering.